Project 3: 3D Object Detection from Lidar Point Clouds

Deep Learning for Autonomous Driving
Team ID 18
Ivan Alberico, Nicola Loi
May 17, 2021 - July 2, 2021

Problem 2 - Taking Matters Into Your Own Hands

Abstract

The network exploited in Problem 1 performs acceptable mAP scores, even for the hard difficulty, but it still remains a baseline network with a simple structure. It does not exploit several simple additional approaches that could increase its performance. Here we will evaluate some modifications and/or additions to the baseline network, to find which methods are able to enhance its car detection capability. At the end, our final network exploits a change in the reference frame of the pooled points, an additional distance feature, and data augmentation approaches. The final network has a noticeable increase in the mAP scores with respect to the baseline method, decreeing the success of the improvements.

Introduction

The greater shortcoming of the baseline network is the different magnitude of the values of the point coordinates. Since not only the extracted local features but also the absolute point coordinates are fed to the Pointnet++ network backbone, the network is not invariant to the absolute position of the points. The goal of the second stage of the network is not to localize a car in the scene, but to classify as car or non-car the given RoI of the point cloud: it only receives as input small individual sections of the point cloud. From the points within these individual RoI, the network has to classify them and regress a bounding box around them. Since the regression and classification of the region of interest are not linked to the absolute position of the points but only to their relative position within the RoI, the network could be improved to be robust and invariant to the absolute position of the pooled points.

Apart for this specific modification, the other experiments performed were more general. We started the testing trying a new classification loss, to see if the background/foreground proposal ratio could be better exploited, and a new regression loss, to try input for the L1-Smooth loss different from the basic difference between the predicted and the ground truth box parameters. Finally, multiple data augmentation approaches are exploited, to increase the variety of the training set so the network could be trained with more different samples and be more robust against car variations.

Method

First, two different losses are evaluated, to find out if the baseline network will experiences an initial improvement with a different loss. For the classification loss, we tried the focal loss [1][2], which is a useful implementation when the background/foreground samples ratio is very high. Its best use should be in the first stage, not the second stage of the network where we already perform a sampling to avoid an high background/foreground ratio, but we preferred to test it anyway to evaluate it. It

was implemented on top of the BCE loss already computed in Problem 1, by premultiplying it with a factor depending on the confidence score of the predictions, which changes according to the class we are taking into account, as in the following:

$$L_{focal}(x_t, y) = -\alpha_t (1 - x_t)^{\gamma} \log x_t = -\alpha_t (1 - x_t)^{\gamma} \,\mathrm{BCE}(x, y) \tag{1}$$

$$\text{where} \quad x_t = \begin{cases} x & \text{if } y = 1 \\ 1 - x & \text{if } y = 0 \end{cases}$$

Instead, for the regression loss we tried the implementation shown in [4], where the rotation loss stays the same, but in the location loss the errors in the predictions are normalized by the predicted dimension of the box, and in the size loss the logarithm of the proportion between the prediction and the ground truth is used instead of their difference, as in the following way:

$$L_{location} = \mathrm{smoothL1}\left[(\Delta x, \Delta y, \Delta z), (0,0,0)\right] \tag{2}$$

$$L_{size} = \mathrm{smoothL1}\left[(\Delta h, \Delta w, \Delta l), (0,0,0)\right] \tag{3}$$

$$L_{rotation} = \mathrm{smoothL1}\left[\Delta\theta, 0\right] \tag{4}$$

where $\Delta\mathrm{x} = \frac{x_{target} - x_{pred}}{diag}$, $\Delta\mathrm{y} = \frac{y_{target} - y_{pred}}{diag}$, $\Delta\mathrm{z} = \frac{z_{target} - z_{pred}}{h_{pred}}$, $diag = \sqrt{l_{pred}^2 + w_{pred}^2}$;
$\Delta\mathrm{l} = \log(\frac{l_{target}}{l_{pred}})$, $\Delta\mathrm{w} = \log(\frac{w_{target}}{w_{pred}})$, $\Delta\mathrm{h} = \log(\frac{h_{target}}{h_{pred}})$; $\Delta\theta = \theta_{target} - \theta_{pred}$.

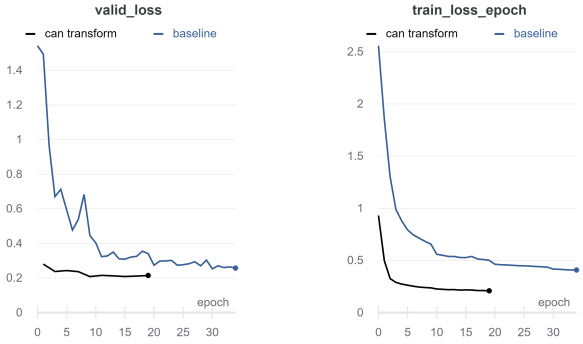
To make the network invariant to the absolute position of the pooled points, every RoI must be transformed to a different reference frame before feeding it to the network, so that the previous original position of the points lose importance with respect to their relative position. This is achieved with a canonical transformation, i.e. translating and rotating the pooled points to the frame whose origin is the center of the proposal and whose axes are aligned with the box edges, as suggested in [3]. Following the paper, we also added the distance to the origin in the original frame ($d_i = \sqrt{x_i^2 + y_i^2 + z_i^2}$) as a feature for the points, to retain the information about the point distance to the LiDAR, since the density and the number of points in each proposal decrease with the distance. The distance is normalized dividing it by 85, which is the maximum distance of a proposal. This method is implemented in the *roi_pool()* function, where the points are already transformed to check if they are within the enlarged box. Before, these new coordinates were only used for the checking, but now they are also used as the coordinates of the pooled points.

The data augmentation is performed in two main ways. The *x,y,z* coordinates of the points of each pooled RoI are individually randomly scaled by a factor uniformly distributed in the range [0.9, 1.1]. In other words, each pooled RoI is not scaled by a unique factor in all directions, but each direction is scaled by its own factor, to be even more robust against cars with mixed proportions. The second augmentation method is to randomly flip with a 50% chance the pooled RoI in the canonical frame around the x-axis, i.e. inverting the z coordinate of each point, to exchange the right side of the car with the left side. In addition, each each point is separately disturbed by a random uniformly distributed noise in the range [-0.01, 0.01] m. Each point coordinates are individually disturbed, to simulate the noise of the LiDAR sensor in a very simple formulation. With respect to the scaling and the flipping augmentation approaches this method is of secondary importance, nonetheless it could help the network in being more robust against the slight position fluctuations of the points due to the noise of the sensor. The second stage network takes as input not only the coordinates of the points coordinates but also their features; since the data augmentation is changing the relative positions of the points, it could create contrasts with the information delivered by the features, since they were extracted in the first stage from the original point coordinates.

Results

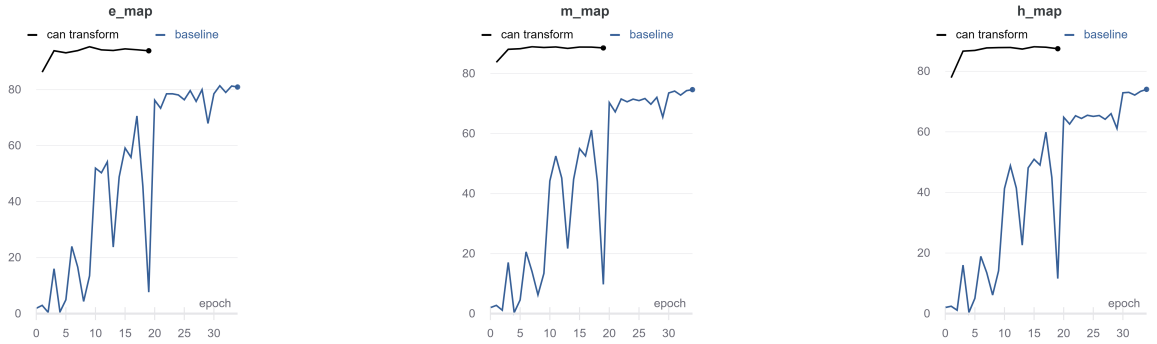
The mAP scores of the baseline network on the validation set are `easy:` 80.93, `medium:` 74.68, `hard:` 74.04, while on the test set are `easy:` 79.167, `medium:` 72.43, `hard:` 71.34. All the experiments performed are evaluated on the validation set; only for the final network the mAP scores will be evaluated also on the test set.

The two different losses evaluated did not bring any improvements to the final performance of the network. With the focal classification loss, the final mAP scores are `easy:` 75.60, `medium:` 71.48, `hard:` 69.54, worsening the results by a few points, especially for the `easy` mAP. As thought, having already made a background/foreground sampling, the focal loss has not given added value to the training. The training with the modified regression loss was instead very unstable, due to the divisions present, even tough was added an epsilon of 0.01 to the denominators to avoid possible division by zero. Each of the seven component of the loss was clamped to a max value of 4 to be more stable, but since in the first epochs the loss was not decreasing and the mAP scores were in the order of 1e-3, hence practically zero, the training was aborted and decreed a failure.



(a) (b)

Figure 1: Comparison of the losses of the network between the baseline and after implementing the canonical transformation: (a) training loss, (b) validation loss.



(a) (b) (c)

Figure 2: Comparison of the mAP scores between the baseline and after adding the canonical transformation: (a) easy, (b) medium, (c) hard.

The canonical transformation, instead, goes in a different direction, since it does not affect how the loss is computed but it applies directly on the points. Introducing the canonical transformation into the pipeline increasingly boosted the performance of the network, achieving high mAP scores in a shorter amount of time. In fact, already after a few epochs, the mAP scores achieved by the network on the validation set were way higher than those from Problem 1 after the whole training of 35 epochs. The mAP scores obtained after 10 epochs are `easy:` 93.877, `medium:` 88.595, `hard:` 87.453. Since no improvements were recorded afterwards and the validation loss was stopping decreasing, we can

consider the network trained after just 10 epochs, with very high results (Figure 1 and Figure 2). Since the validation loss and the mAP scores were shortly saturated and the training loss witnessed a very fast decreasing at the beginning with a slow decreasing afterwards, it could seem that the network was learning too fast. It is then tested a diminishing of the learning rate, from the original 0.004 (4e-3) to 0.0008 (8e-4), i.e. 1/5 of the original. Even tough the mAP scores, the validation and the training loss have now a more smooth increasing/decreasing, their final values are practically the same as for the original learning rate. Since there were no immediate visible improvements, we kept the original learning rate.

As the canonical transformation significantly contributed to improving the results, all the following changes in the network were tested on top of that. We tried adding the point distances to the features in order to include also some depth information while supervising the training. But it did not bring substantial changes, reaching the following mAP scores after 10 epochs: `easy:` 94.645, `medium:` 88.799, `hard:` 87.589.

One of the last modification that was tested was introducing data augmentation while training the network. Introducing data augmentation did not bring improvements of the network, even though the results were somehow comparable to those obtained including the distance in the features. In fact, the obtained mAP scores on the validation set are `easy:` 94.63, `medium:` 88.26, `hard:` 87.28.

Even though the additional distance feature and the data augmentation did not improve the scores of the network, we decided to keep them in the pipeline, because they don't worsen it anyway and the training time is unchanged. Having them could benefit the detections in certain situations and scenes, even though in the current validation set it was not observed any improvement. The mAP score on the test set for the final network with the canonical transformation, the distance feature and the data augmentation after 10 epochs are `easy:` 93.48, `medium:` 88.66, `hard:` 88.21. The final training time is 2 hours and 10 minutes, while the training time of the baseline was 6 hours and 25 minutes (both times include validation at each epoch)

Discussion

Without the shadow of a doubt, the major contribution of the results that have been achieved comes from applying the canonical transformation to the pooled points of each bounding box proposal of the first stage. In general, the reason why the canonical transformation helps obtaining better results is that it eliminates variations among different 3D proposals in terms of position and orientation in the scene. Even though the network with the distance feature and the data augmentation performed similarly, it is chosen to keep these additions. Data augmentation represents a regularization technique and, in general, it is advisable to use it, since it attempts to make the network less prone to overfit. Moreover, it is very simple and efficient to implement, and also it makes the network more robust to variations in the input. Also the distance feature is a very fast yet efficient modification that integrates depth information in the training process, by putting together additional local spatial features to the global features previously computed by the first stage. Finally, our network not only outperformed the baseline but it needs only one third of the time to train, another astonishing improvement.

On the other hand, the modifications made on the loss were not as successful as the ones just discussed. Both the focal loss and the modified regression loss from VoxelNet [4] did not bring any enhancement to the network, or rather, they performed even worse or made the whole process unstable.

Given the hardware problems encountered, it was not practicable to test other possible improvements and/or to better test our improvements. Many other network modifications should be experimented in future works. For instance, remodelling of the PointNet++ layers and of the final classification and regression branches, testing different number of layers and channels. The PointNet++ backbone network could also be completely abandoned in favor of other models.

References

[1] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *CoRR*, abs/1708.02002, 2017. URL `http://arxiv.org/abs/1708.02002`.

[2] S. Shi, X. Wang, and H. Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *CoRR*, abs/1812.04244, 2018. URL `http://arxiv.org/abs/1812.04244`.

[3] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li. PV-RCNN: point-voxel feature set abstraction for 3d object detection. *CoRR*, abs/1912.13192, 2019. URL `http://arxiv.org/abs/1912.13192`.

[4] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017. URL `http://arxiv.org/abs/1711.06396`.